INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA
ESCUELA DE INGENIERÍA Y GESTIÓN

Estudio de similitudes sobre los algoritmos de triangulaciones basadas en markerless tracking

AUTORES: Lucía Tay (Leg. Nº 56244)
Nicolás Andrés Paganini (Leg. Nº 54321)

TUTORA: Marcela Alejandra Guerrero

REPOSITORIO: https://bitbucket.org/itba/pf-markless3d-realtime

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA EN INFORMÁTICA

Lugar: Ciudad Autónoma de Buenos Aires

Fecha: 02/09/21

Índice

1. Introducción

Motion capture refiere al proceso de capturar el movimiento en tres dimensiones de actores humanos en el mundo real y luego aplicar esos movimientos a avatares en una escena virtual [1]. Para lograr esto, es necesario contar con más de una cámara grabando al actor en cuestión y herramientas para detectar las partes del cuerpo en la imagen capturada. Usualmente, se utilizan cámaras específicas acompañadas de un *software* capaz de identificar marcas y que el actor utilice un traje especialmente diseñado con marcas detectables por estas cámaras. Estos trajes permiten que el *software* pueda estimar la posición del cuerpo de la persona a partir de las imágenes capturadas por las cámaras en cada *frame*. Esto a su vez, hace posible la reconstrucción tridimensional de la persona que lleva el traje. A este proceso de usar dos o más imágenes de cámaras para reconstruir a la persona en 3D se le llama triangulación.

El proceso de triangulación es uno de los pasos más importantes del *motion capture*, dado que es la etapa en la cual las diferentes capturas tomadas por las distintas cámaras se juntan en un sistema de coordenadas único. Por esta razón, el desarrollo de una triangulación eficiente en términos de velocidad y precisión es fundamental. Este proceso puede ser muy costoso computacionalmente, dado que se deben procesar las imágenes 2D de todas las cámaras para todos los *frames* de un video y generar una reconstrucción en 3D por cada uno de ellos. Adicionalmente, el traje de captura de movimiento suele ser un equipamiento costoso. La suma de estas dos problemáticas aumenta la barrera de entrada a aquellos que quieran adentrarse en el tema.

Así como el proceso de identificación de las marcas se puede realizar de manera física mediante el uso del traje de captura recién mencionado, también puede ser realizado de manera digital (en la cual no se requiere ningún *hardware* específico). La versión digital ocurre a través del reconocimiento de marcas, o *markers*, a través de redes de aprendizaje profundo [2]. Este sistema se conoce como *markerless human tracking* [3]. Dado que se descarta el uso de un traje de captura de movimiento, la barrera de entrada introducida por el alto costo del equipamiento desaparece. Sin embargo, la detección de los *markers* es un proceso difícil, en términos de velocidad y precisión, sin la certeza que brinda el traje físico. Es por esto que el paso de triangulación se torna más importante, dado que tiene que cubrir la velocidad y precisión perdida en pasos anteriores.

El objetivo de este proyecto es comparar distintos métodos y *frameworks* de triangulación. En el contexto de *computer vision* el proceso de triangulación refiere a la determinación de la posición de un punto en el espacio tridimensional a partir de su proyección en dos o más imágenes en dos dimensiones. Lo que se busca triangular son partes del cuerpo identificadas en imágenes a través de *markers*. En primer lugar, para la detección digital de *markers*, se utilizó el *framework* de *deep learning* para estimación de poses *DeepLabCut* [4]. Este permitió entrenar un modelo que, a partir de grabaciones tomadas de varias cámaras en simultáneo en una sala de captura de movimiento, rastrea puntos de interés en humanos sin el uso de trajes especiales. De este *framework* se espera, a su vez, que el modelo entrenado permita resolver oclusiones parciales de *markers*. Luego, tomando estos *markers* y utilizando la información de calibración de las cámaras [5] para las grabaciones dadas, se tomaron los puntos 2D capturados por cada cámara y, finalmente, se proyectaron los *markers* en 3D en el proceso de triangulación.

En este informe se comparan cuatro algoritmos distintos de dicho proceso con la intención de reducir tiempos de procesamiento y aumentar la precisión de las triangulaciones previamente utilizadas. A través de nuestro proyecto se plantean dos triangulaciones superiores a las ya existentes. Estas reducen significativamente el tiempo de procesamiento del paso de triangulación, además de incrementar la precisión con la cual se forma cada reconstrucción en 3D. Por último, algunos de estos algoritmos permiten la aplicación de filtros de suavización que resultan en una triangulación con aún menos irregularidades. Por lo tanto, también se estudiarán los distintos resultados obtenidos luego de dicha aplicación.

2. Consideraciones y Contextualización

Este trabajo comenzó en marzo del 2020 y finalizó en agosto del 2021. La idea original de nuestra investigación consideraba la asistencia a la sala de captura de movimiento de la sede del Distrito Tecnológico (sala ubicada en Parque Patricios a cargo de Marcela Guerrero, tutora del presente trabajo) del ITBA para encarar el proyecto de análisis de *markerless tracking* en tiempo real.

Poco tiempo después del comienzo de nuestro proyecto, el 11 de marzo, la Organización Mundial de la Salud anunció la pandemia del COVID-19, con la repercusión siguiente siendo el comienzo del aislamiento social preventivo y obligatorio en todo el territorio argentino el 19 de marzo, de duración original de casi dos semanas.

Con el comienzo de la cuarentena, el equipo se vio forzado a adaptar el proyecto sobre la marcha debido a la incertidumbre de cuándo se podría volver a la sala de captura. Al no contar con acceso a la sala, se decidió darle al proyecto un enfoque más teórico. Esto conllevó la dedicación del tiempo del proyecto a la investigación del marco teórico y posibles mejoras al proyecto antecesor de los alumnos Lóránt Mikolas y Agustín Calatayud.

Mientras la pandemia continuaba, la cuarentena se extendió para acompañar el incremento de casos. A partir de las medidas tomadas de distanciamiento social, se comenzó a descartar objetivos que contaban con el uso de la sala de captura y se viró hacia objetivos que se pudiesen realizar de manera remota. Finalmente se resolvió trabajar en una comparación de distintos algoritmos e implementaciones de triangulación.

Si bien en enero del 2021 la universidad expresó la posibilidad de visitar la sala y realizar distintas mediciones, impusieron una limitación de acceso de una persona por vez. Por lo tanto, dicha oferta fue rechazada casi inmediatamente ya que ninguno de los miembros del equipo tenía experiencia con el funcionamiento de la sala de captura.

Finalmente, dadas las circunstancias de la pandemia mencionadas, se desea destacar que el proyecto fue realizado en su totalidad de manera remota, utilizando las computadoras personales tanto para la investigación y aprendizaje teórico, así como el posterior análisis sobre los métodos de triangulación elegidos.

3. Antecedentes

Como fue mencionado en la sección anterior, los objetivos de este proyecto tuvieron que ser repensados para ajustarse al contexto de la pandemia. Sin embargo, llegar a la decisión de comparar distintos algoritmos de triangulación no fue un proceso fácil. De hecho, se consideraron varias otras posibilidades que luego fueron descartadas por razones varias. En esta sección se detallarán algunas de ellas.

En primer lugar, se examinó la posibilidad de estudiar e implementar distintos filtros y optimizaciones de manera tal de reducir la cantidad de errores obtenidos luego del proceso de triangulación. En particular, se estudió un método llamado *Random Sample Consensus* (RANSAC), el cual sirve para detectar *outliers* [6][7]. El algoritmo consiste en tomar un *set* de datos arbitrario de todo el conjunto de datos disponible y determinar si éste se ajusta a algún patrón predeterminado. Este proceso se lleva a cabo de manera iterativa y luego se obtiene el conjunto de datos que contiene la mayor cantidad de *inliers*. El problema que surgió cuando se quiso llevar este método a la práctica fue que resultaba imposible determinar el patrón al cual se debían ajustar los datos. Dada la naturaleza impredecible de los *datasets* con los que se contaba, y sumado a la imposibilidad de realizar nuestras propias capturas y mediciones, desafortunadamente esta idea debió ser descartada.

Otro tema estudiado fue el de las homografías. Se trata de transformaciones que determinan una correspondencia entre dos planos bidimensionales. Por lo tanto, su uso queda limitado a los casos en los que se cuenta con múltiples cámaras apuntando a un mismo plano, o bien escenas capturadas con una sola cámara rotada sobre su propio eje [8][9]. Sin embargo, ninguno de estos casos coincidía con los datos con los que se contaba, ya que se disponía de distintas cámaras que apuntaban a una persona (es decir, un objeto tridimensional, y no un plano), y por dicha razón se tomó la decisión de dejar de lado está investigación.

Luego de decidir que el proyecto iba a comparar triangulaciones, se investigó varias triangulaciones que se podían usar. Entre ellas, la triangulación *Point Streams* [10] fue evaluada como posible opción para comparar. Este método utiliza el flujo de puntos creado por un *scanner* láser para mapear un objeto a una computadora con una precisión sorprendentemente alta en poco tiempo. Sin embargo, está optimizado solo para objetos estáticos. El proceso de pasar de un objeto a una imagen 3D no es costoso si se trata de un solo objeto en un momento dado, pero si se extrapola esto a la grabación constante *frame* a *frame* del objeto y sus movimientos, éste se

vuelve más y más ineficiente a medida que aumentan. Debido a que nuestro proyecto siempre trabajó con los videos que ya teníamos de la sala de captura, decidimos dejar de lado este método de triangulación.

4. Marco Teórico

a. Modelo de proyección en perspectiva

El modelo de cámara en perspectiva [8] describe la correspondencia entre las coordenadas de un punto en el mundo *W* y los puntos capturados en la imagen *I* mediante la siguiente ecuación:

$$u = P^W X \qquad (1)$$

Donde *u* representa las coordenadas del píxel, *P* es la transformación, y *X* son las coordenadas del objeto en el mundo.

En la siguiente figura se puede apreciar una representación gráfica de dicho modelo. Se observa la cámara *C* (la cual cuenta con su propio sistema de coordenadas), el objeto *X,* y el plano de la imagen *I*, donde se proyecta la imagen del objeto de manera bidimensional [11].

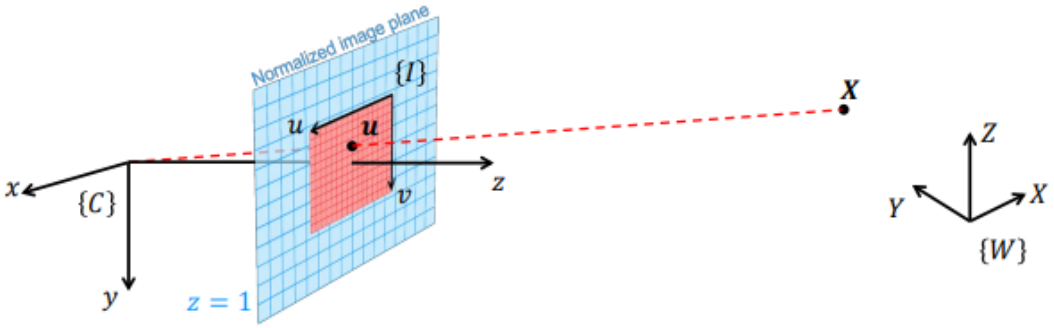

Figura 1: Modelo de cámara en perspectiva

En el contexto de la fotogrametría, a la transformación *P* se la conoce también como matriz de proyección y se puede representar de la siguiente manera:

$$P = KR[I_3| - T] \qquad (2)$$

Donde *K* representa la matriz intrínseca [12] de la cámara, y *R* y *T* son los parámetros extrínsecos [13].

Cuando hablamos de parámetros intrínsecos nos referimos a aquellas características "internas" de la cámara, que son expresadas a través de la matriz *K*:

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

Donde f_x y f_y representan la distancia entre la cámara y el plano de la imagen (distancia focal), x_0 e y_0 la distancia del punto principal y *s* la distorsión del eje principal. Por otro lado, los parámetros extrínsecos refieren a aquellos que relacionan la cámara con la escena y se representan a través de la matriz de rotación *R* y la matriz de traslación *T*:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix}, T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \quad (4)$$

La rotación *R* representa la manera de rotar el sistema de coordenadas del mundo de manera tal que resulte en la misma orientación que el sistema de coordenadas de la cámara, mientras que la traslación *T* expresa la posición del origen de coordenadas del mundo de acuerdo al sistema de coordenadas de la cámara.

Por lo tanto, combinando todas las ecuaciones, la ecuación original (1) se calcula de la siguiente manera:

$$u = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_1 \\ 0 & 1 & 0 & -t_2 \\ 0 & 0 & 1 & -t_3 \end{bmatrix} X \quad (5)$$

b. Triangulación

Como bien fue mencionado anteriormente, el proceso de triangulación consiste en determinar la posición de un objeto en el espacio partiendo de sus coordenadas bidimensionales conocidas en las imágenes proyectadas por cada cámara. [14]. Para ello es necesario conocer los parámetros extrínsecos e intrínsecos de cada cámara, según su definición en las ecuaciones 3 y 4.

Cada punto en una imagen corresponde a una recta en 3D, es decir todos los puntos en dicha recta se proyectan al mismo punto en el plano de proyección. Idealmente, si se logra hacer una correspondencia entre puntos de distintas cámaras, entonces la posición tridimensional del objeto en cuestión queda determinada por la intersección de las rectas. Se puede apreciar una representación gráfica de este proceso en la siguiente figura:

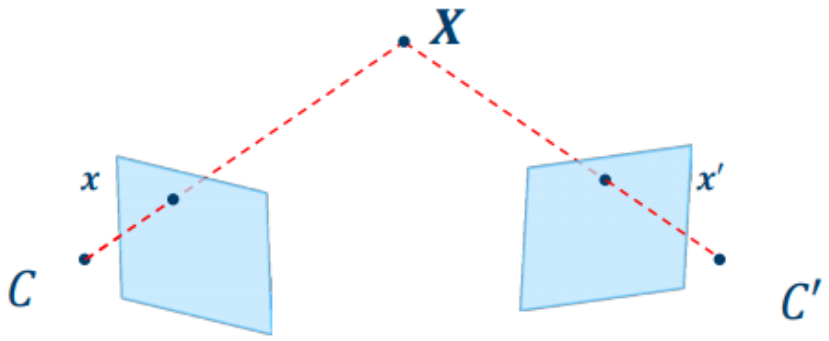

Figura 2: Determinación del punto en 3D mediante la intersección de las rectas

Sin embargo, suele pasar que las proyecciones bidimensionales no son lo suficientemente precisas. Por lo tanto, puede suceder que las líneas generadas por dos pares de puntos correspondientes no resulten en una intersección en el espacio. Entonces el problema de la triangulación se convierte en determinar el punto óptimo del objeto en 3D. En la sección de "Implementaciones" se estudiarán cuatro distintos algoritmos que resuelven dicho problema.

5. Preprocesamiento

a. DeepLabCut

DeepLabCut (*DLC*) es una herramienta para estimar la pose de animales sin uso de marcas físicas a la hora de rastrear movimiento. Esta herramienta es *open source*, es decir, es libre y gratuita, y se puede usar para determinar y seguir el movimiento de animales mientras desarrollan diferentes tareas. Recientemente, *DLC* agregó una red capaz de detectar seres humanos.

El uso de *DLC* en el presente proyecto fue un paso previo pero esencial para poder llevar a cabo las triangulaciones. Este paso se llevó a cabo siempre, sin importar cual fuere el método de triangulación aplicado.

Para correr *DLC* y obtener las predicciones de los *markers*, debimos utilizar la red neuronal *ResNet 101*, una red provista por *DLC* para seguimiento de humanos. Este cuenta con 14 *markers* para detectar distintas partes del cuerpo humano y su uso es sencillo dado que está integrado en una única función, *create_pretrained_project*. Esta función toma el nombre del proyecto, el nombre del autor, la ubicación de los videos a analizar, el modelo a utilizar (en nuestro caso, el *full_human* [15], por el uso de todo el cuerpo de la persona en la grabación disponible), la ubicación del proyecto en la computadora, la opción de copiar los videos con los *markers* o reemplazar los ya existentes y el formato del archivo video. El resultado de correr la función es un *.csv* por cada cámara con las coordenadas de cada *marker* 2D en cada uno de los *frames* de la grabación.

b. Sala de captura

Uno de los elementos esenciales en el proceso de triangulación es la escena que efectivamente se busca triangular. Para ello es necesario capturarla mediante el uso de cámaras, idealmente más de una. En particular, para este proyecto se usaron escenas que fueron filmadas en el laboratorio de captura del ITBA que se encuentra en la Sede Distrito Tecnológico.

La sala de captura cuenta con ocho cámaras *Flex 3 Optitrack*, ubicadas de manera circular. Además, se dispone de varios trajes de captura *Optitrack* para usar en conjunto con el software *Motive* para realizar tanto el rastreo de los *markers* de dichos trajes como la consiguiente triangulación correspondiente. En la siguiente fotografía (cortesía del departamento de

Comunicación Institucional) se puede apreciar tanto la disposición de las cámaras, así como los trajes de captura recién mencionados:

Figura 3: Sala de captura de la Sede Distrito Tecnológico

Las cámaras *Flex 3 Optitrack* permiten realizar capturas de imágenes, las cuales en conjunto con el programa *Motive* ofrecen procesamiento y seguimiento de movimiento. Pueden capturar hasta cien *frames* por segundo con una resolución de 0.3MP (640 x 480). Las imágenes que se obtienen como resultado son en escala de grises [16].

Figura 4: Cámara *Flex 3 Optitrack*

c. Calibración de las cámaras

Para la calibración de las cámaras se armó un escenario en el laboratorio de toma de captura con las 8 cámaras disponibles a su alrededor. A partir de este arreglo, se obtuvieron los valores

de los parámetros extrínsecos e intrínsecos mediante el uso de la herramienta 3D-Pose [5] del alumno Braulio Sespede. En este se utilizó herramientas como OpenCV 4.1.0 con Contrib Modules (aruco), OptiTrack Camera SDK 2.1.1 y OpenPose 1.5 para obtener los valores base de la ubicación de las distintas cámaras, los cuales son requeridos para la triangulación.

La herramienta 3D-Pose calcula las posiciones y los ángulos de las cámaras a partir de la posición del tablero *ChArUco*, un tablero que se coloca en el suelo para calibrar las diferentes cámaras a partir de las medidas del mismo, las cuales son conocidas de antemano.

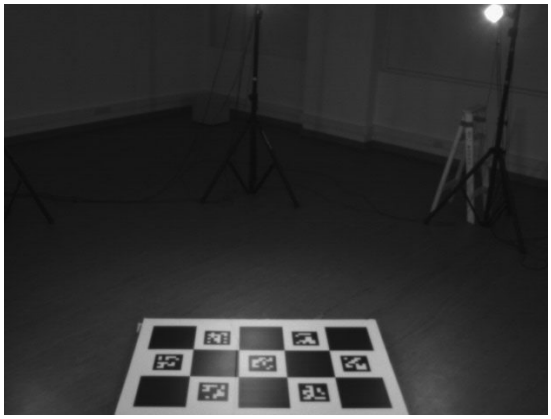

Figura 5: Escena de calibración con el *ChArUco board*

A partir de las grabaciones obtenidas de filmar el tablero se calculan las posiciones de las cámaras relativas al tablero, para así establecer las coordenadas del mundo en el que se encuentran todas las cámaras. Una vez que se consigue dicha información, el programa devuelve dos archivos *.json* con los parámetros necesarios para armar las matrices extrínsecas e intrínsecas definidas en las ecuaciones 3 y 4. A su vez estos archivos son usados como entradas en las implementaciones de los algoritmos de triangulación descritos en la siguiente sección.

6. Implementación

a. Triangulación lineal (DLTdv)

Considerando la transformación descrita anteriormente en la ecuación 5, cada una de las cámaras aporta un sistema de ecuaciones que por sí solas no son determinantes. Sin embargo, si se combinan las ecuaciones de dos o más cámaras para el mismo punto, se logra obtener un sistema sobredeterminado de ecuaciones lineales homogéneas. Luego, se aplica el método de cuadrados mínimos para hallar el resultado óptimo de la triangulación [17].

La implementación de este algoritmo se encuentra en *DLTdv*, un programa desarrollado por Tyson Hedrick y los integrantes de su laboratorio, donde estudian la locomoción y el mecanismo de vuelo de distintos animales [18]. Cabe destacar que es la única implementación estudiada que fue desarrollada en *Matlab*, haciendo uso de las librerías y funciones matemáticas que ofrece el mismo.

Para hacer uso de este programa fue necesario traducir los archivos *.csv* que se obtuvieron como resultado de *DeepLabCut* en un sólo archivo que contuviese las coordenadas 2D de todos los *markers* en cada cámara. Al iniciar el programa es necesario indicar tanto la cantidad de cámaras con las que se está trabajando, así como la matriz de coeficientes de *DLT* (la cual fue definida como *P* en la ecuación 2). Luego, se carga el archivo de coordenadas descrito anteriormente y se le indica al programa que debe realizar la triangulación mediante la función "*Recompute 3D points*". Por último, se exportan los resultados a un archivo *.csv* donde se pueden observar las coordenadas 3D.

Vale la pena mencionar que si bien ya se disponía de las coordenadas 2D de todos los puntos, *DLTdv* cuenta con las funciones necesarias para obtener esos datos. Funciona de manera similar a *DeepLabCut*, donde se van etiquetando los *markers* en cada *frame* para que luego el programa pueda realizar estimaciones y predicciones sobre la posición de cada uno de ellos. Además, cuenta con un programa auxiliar para realizar la calibración de las cámaras y de esa forma poder obtener la matriz de coeficientes *DLT* necesaria para realizar la triangulación.

b. Anipose

La triangulación de *Anipose* [19] es una triangulación desarrollada por Pierre Karashchuk y Katie Rupp de la Universidad de Washington, Estados Unidos. Esta cuenta con diferentes herramientas para la triangulación de puntos observables en distintas cámaras.

Originalmente desarrollado para el uso de rastreo de animales, *Anipose* estima la pose de animales tomados a partir de distintas cámaras para proyectar al animal en el mundo 3D. Para esto utiliza un archivo de configuración general para habilitar las distintas funciones y filtros que ofrece. Adicionalmente, *Anipose* utiliza *Aniposelib*, una herramienta desarrollada por Pierre Karashchuk para la administración de las cámaras durante la triangulación [20]. Ambos proyectos están desarrollados en *Python* y utilizan la librería *OpenCV* [21].

En nuestro caso, no se utilizó *Anipose* para poses de animales, sino para humanos. La conversión fue transparente dado que de base, *Anipose* solo hace la triangulación de los *M* markers de las *N* cámaras. Luego, junto al archivo de configuración propuesto, calcula las coordenadas de dónde se encuentra lo observado. Esta generalidad natural es la que permitió el pasaje de animales a humanos de manera transparente, además de dar independencia de la cantidad de *markers* y cámaras necesarias para la triangulación (siempre y cuando la cantidad de cámaras sea mayor a 1).

Para integrar la triangulación de *Anipose* al proyecto *Markerless Human Tracking* se realizaron cambios al proyecto original elaborado en *Python* por Agustín Calatayud y Lóránt Mikolás. Primero se instaló *Anipose* para poder correrlo como *package* de *Python* independiente. Una vez instalado, se requirió configurar el uso mediante un archivo de configuración *config.toml* y se requirió configurar las posiciones de las cámaras mediante un archivo de configuración *calibration.toml*. Este último hizo uso de los valores de las extrínsecas e intrínsecas obtenidas previamente en el paso de calibración de las cámaras. Una vez que ambos archivos de configuración estaban disponibles en el proyecto *Markerless Human Tracking*, se modificó el archivo *point_helpers.py* para poder designar la ubicación de los *.csv* elaborados por *DLC* y triangular usando la función *triangulate* propuesta por el paquete *Anipose*. La función toma el archivo de configuración general, la carpeta que contiene al archivo de calibración de las cámaras, la carpeta que contiene a los videos de las cámaras, la carpeta que contiene los *.csv* elaborados por *DLC*, el listado de las cámaras, y el nombre que se le quiere dar al *.csv* con los

frames triangulados. Ésta procesa los datos y devuelve un *.csv* con la posición (x,y,z) de cada *marker* en cada *frame* en el sistema de coordenadas establecido por la calibración de las cámaras.

Adicionalmente, *Anipose* ofrece múltiples opciones de configuración, entre las cuales figura la opción de aplicar un filtro 3D a fin de corregir posibles errores que puedan haber surgido a partir del proceso de triangulación. Adicionalmente, tiene la posibilidad de definir restricciones sobre el grado aceptable de saltos entre *frames*, independientemente de si el filtro está activo o no, dónde, si no lo es, descarta la triangulación del *marker* para el par *frame*-cámara.

c. Triangulación Stereo (OpenCV)

La triangulación stereo está basada en la geometría epipolar, la cual está fundada en la relación geométrica entre dos cámaras. Idealmente consiste en un *setup* de dos cámaras paralelas, ubicadas a lo largo de un eje y con la misma orientación, como se puede observar en la siguiente figura:

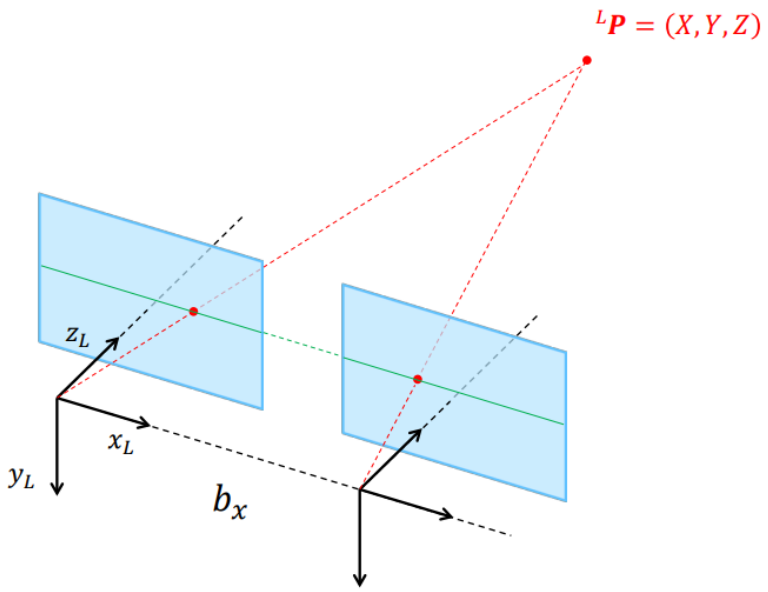

Figura 6: Configuración ideal de un sistema de cámaras stereo

La línea que conecta los centros de las lentes es conocida como línea base o *baseline* y la distancia entre ellas la denominamos *b*, mientras que la línea que une los puntos correspondientes entre imágenes se llama línea epipolar. Si consideramos que ambas cámaras tienen la misma distancia focal, podemos calcular las coordenadas del punto en el espacio mediante las siguientes ecuaciones:

$$Z = f\frac{b_x}{d}$$

$$X = x_L\frac{Z}{f}$$

$$Y = y_L\frac{Z}{f} \qquad (6)$$

Donde f es la distancia focal, b_x es la distancia entre las cámaras, x_L e y_L son las coordenadas del punto en la imagen (en el sistema de coordenadas de la cámara izquierda), d es la disparidad entre el punto observado en ambas cámaras y X,Y,Z son las coordenadas del objeto en el espacio.

Sin embargo, no siempre se puede armar el *setup* de manera tal que coincida con la configuración ideal descrita anteriormente, sino que nos encontramos con cámaras con distintas orientaciones, como es el caso de la sala de captura del ITBA. En este caso, es necesario realizar un ajuste para poder aplicar los conceptos de la geometría epipolar. A dicho ajuste se lo llama rectificación stereo y consiste en re-proyectar los planos de las imágenes a un plano común paralelo a la línea base entre las cámaras.

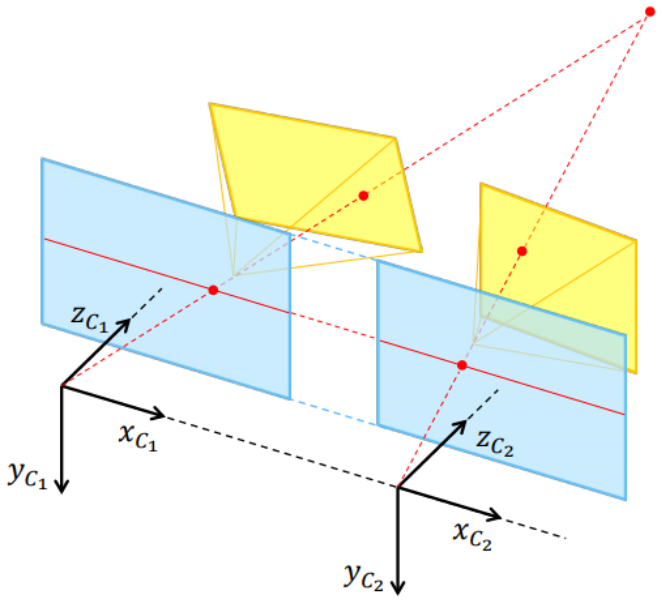

Figura 7: Rectificación stereo

Este método fue implementado por Lóránt Mikolás y Agustín Calatayud como parte de su proyecto final [3][22], utilizando *Python* como lenguaje de programación, y en particular, la librería *OpenCV* que cuenta con métodos para realizar tanto la triangulación como la rectificación stereo [21].

La implementación del método toma un par de cámaras adyacentes. Es decir, dos cámaras frontales adyacentes entre sí o dos cámaras traseras adyacentes entre sí. Como la grabación se hizo a partir de 8 cámaras, se ideó un proceso de identificación de las mejoras cámaras evaluadas, basándose en un umbral arbitrario definido por ellos, tomando el promedio de todos los *markers* para todos los *frames* analizados, por cámara.

A partir de estas dos cámaras se calcula la traslación y rotación relativa de las imágenes para calcular su traslación relativa al sistema de la cámara izquierda. Una vez que se tiene este sistema relativo, se aplica una rotación más para llevar el sistema de referencias al de la cámara izquierda.

Finalmente, con la información obtenida, la matriz de las cámaras, los coeficientes de distorsión y el tamaño de las imágenes se puede aplicar la función de Rectificación Stereo de *CV2*. Esto permite obtener las matrices de rotación y las matrices de proyección de las cámaras para luego utilizar la función *triangulatePoints* de *OpenCV*. El resultado de esta función es la

triangulación de las coordenadas en el sistema de coordenadas de la cámara izquierda. Es por esto que se debe aplicar una última transformación para obtener el resultado final en el sistema de coordenadas del mundo.

Una vez obtenidos los resultados de la triangulación, esta implementación ofrece la posibilidad de aplicar un *Kalman filter*. Este se basa en un modelo físico (movimiento rectilíneo uniformemente variado) para predecir el movimiento del cuerpo y, de esta manera, poder suavizar los puntos obtenidos a partir de la triangulación.

d. Triangulación Multi-Cámara (PyMVG)

A diferencia del método anterior, la triangulación multi-cámara permite usar más de dos vistas para proporcionar más detalles sobre la escena. Más imágenes implica tanto mayor robustez frente a oclusiones así como la posibilidad de brindar mayor confianza a la hora de realizar las correspondencias entre ellas.

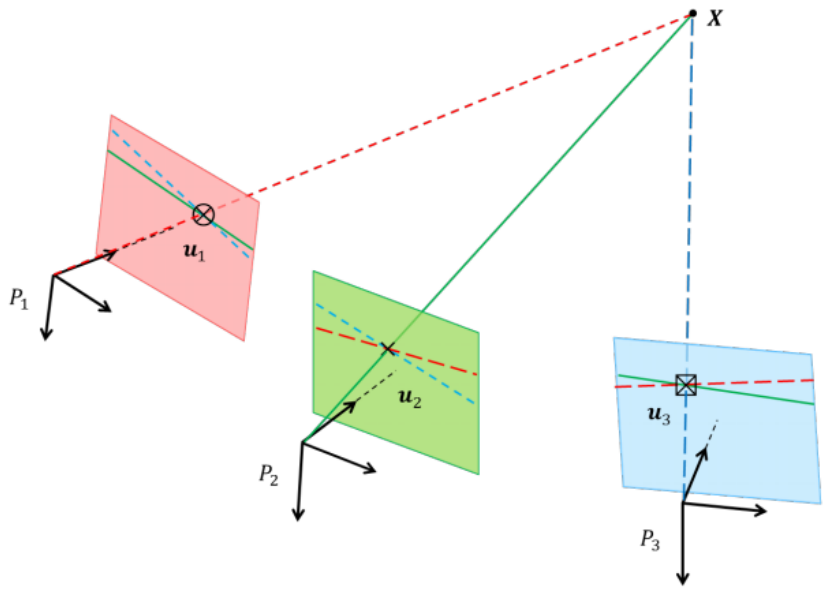

Figura 8: Triangulación Multi-Cámara

La implementación de este método se encuentra en el repositorio *Markerless Human Tracking* [22]. Si bien se encuentra también implementada en *Python,* este método hace uso de las funciones de triangulación provistas por la librería *PyMVG* [23].

El método toma todas las cámaras disponibles y analiza cuáles son las mejores cámaras para triangular cada *marker*, basándose en un umbral arbitrario. A partir de esto, busca que la escena tenga al menos dos cámaras para triangular, dando en promedio cuatro cámaras como entrada. En el raro caso en el que no se encuentren al menos dos cámaras con el valor mínimo del umbral elegido, se disminuye el valor y se analiza a las grabaciones nuevamente, hasta encontrar al menos dos para la escena.

Una vez que se tienen todas las cámaras elegidas, se triangula utilizando la función *find3d* que realiza una triangulación lineal a partir de los datos de las cámaras y la calibración de ellas.

Así como el método anterior, esta implementación también permite aplicar el filtro de *Kalman* para suavizar los resultados.

7. Análisis y Resultados

a. Implementaciones

En primer lugar, se realizó la siguiente tabla a fin de poder comparar las diferentes implementaciones y algoritmos descritos en la sección anterior.

	DLTdv	**Anipose**	**Triangulación Stereo**	**Triangulación Multi-Cámara**
Lenguaje de desarrollo	Matlab	Python	Python	Python
Librerías utilizadas	Linear Algebra (*built-in*)	OpenCV	OpenCV	PyMVG
Input	Archivo *.csv* con los valores de la matriz de proyección + Archivo *.csv* con las posiciones de los *markers* en cada cámara y *frame*	Archivos *.csv* y *.h5* con las posiciones de los *markers* en cada cámara y *frame* + Archivo *config.toml* con información los parámetros extrínsecos e intrínsecos	Archivo *.csv* con las posiciones de los *markers* en cada cámara y *frame* + Archivo *.json* con los parámetros extrínsecos e intrínsecos	Archivo *.csv* con las posiciones de los *markers* en cada cámara y *frame* + Archivo *.json* con los parámetros extrínsecos e intrínsecos
Output	Archivo *.csv* con las posiciones 3D de los *markers* en cada *frame*	Archivo *.csv* con las posiciones 3D de los *markers* en cada *frame*	Archivos *.csv* e *.xyz* con las posiciones 3D de los *markers* en cada *frame*	Archivos *.csv e .xyz* con las posiciones 3D de los *markers* en cada *frame*
Filtro	Ninguno	Optimización de Anipose	Kalman filter	Kalman filter
Año de desarrollo	2020	2019	2020	2020
Documentación	Paper [24], Página web [18], Repositorio [25]	Paper [26], Página web [19], Repositorio [27]	Repositorio[22], Informe académico [3]	Repositorio [22], Informe académico [3]

b. Tiempo de ejecución

En esta sección estaremos analizando los tiempos de ejecución de los distintos métodos de triangulación, incluyendo el tiempo que tardan en aplicar los filtros de suavización en los casos que corresponda. Vale remarcar que para una comparación más efectiva, se tomaron los tiempos únicamente de las funciones que realizan las triangulaciones, dejando de lado toda la parte de procesamiento de archivos.

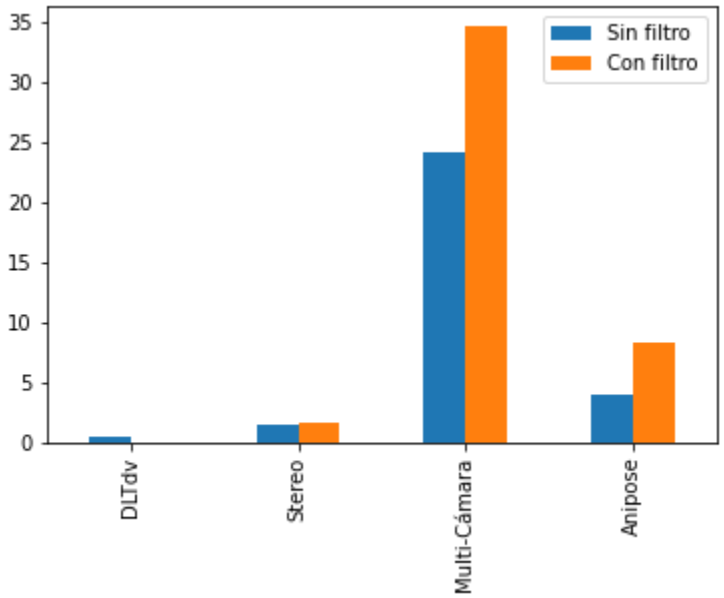

Figura 9: Comparación del *runtime* de los métodos de triangulación

Como se puede observar en la figura 9, la ejecución más rápida fue la triangulación lineal de *DLTdv*, con un tiempo menor a 1 segundo. Esto se debe a que se trata de uno de los métodos más directos de resolver, es decir, no realiza ninguna discriminación a la hora de elegir cuáles cámaras usar, sino que utiliza todas. Además, resulta interesante destacar que *Matlab* se trata de un potente lenguaje matemático, por lo cual es ideal para este tipo de implementación.

Luego siguen la triangulación stereo y la de *Anipose* con unos tiempos de ejecución de 1.5 y 4 segundos respectivamente. Si bien en el caso de la triangulación stereo el tiempo de aplicación del filtro fue casi nulo (la diferencia fue de 0.3 segundos), para *Anipose* resultó en un *runtime* de casi el doble de tiempo que el original.

Por último, observamos que la triangulación multi-cámara tardó considerablemente más que el resto, con un *runtime* de 24 segundos para la triangulación en sí, y aproximadamente 10 segundos para la aplicación del filtro, resultando en un tiempo total de 34 segundos. Creemos que semejante diferencia con respecto a los demás métodos se debe a la necesidad de analizar todas las combinaciones de cámaras posibles para determinar el mejor resultado.

c. Outliers y error promedio

Para realizar un análisis cualitativo sobre los resultados obtenidos de las distintas triangulaciones fue necesario implementar un programa que pudiese procesar los distintos archivos de *output* para transformarlos en un archivo que fuese soportado por algún software de visualización. En particular, se optó por usar archivos *.xyz* ya que éstos pueden ser cargados en un software llamado *Ovito* [28], el cual es capaz de mostrar gráficamente los puntos triangulados.

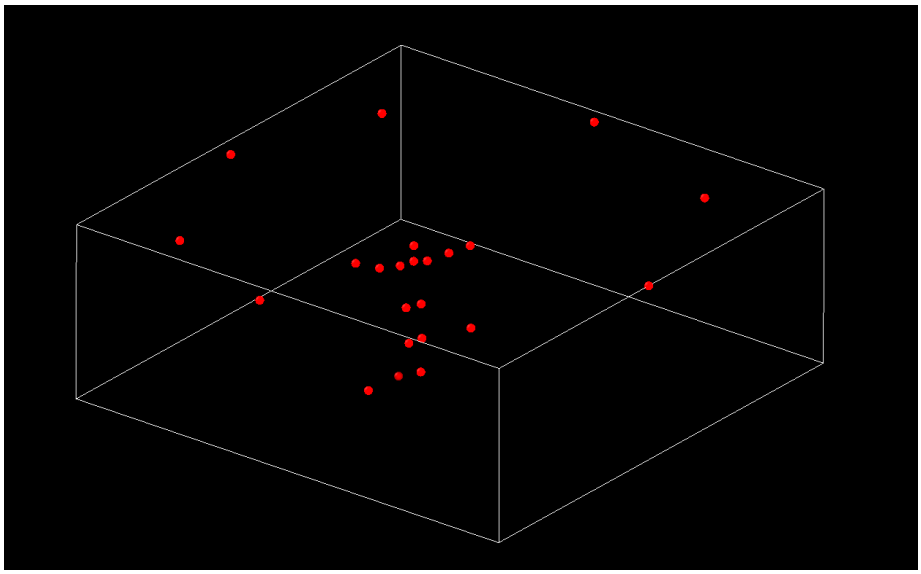

Figura 10: Visualización de los puntos triangulados en *Ovito*

Esto fue especialmente útil para poder sacar conclusiones intuitivas sobre los distintos métodos, así como también una fácil identificación de posibles errores. En especial se notó la falta de precisión de la triangulación stereo, donde los *markers* realizan varios "saltos" entre cada *frame*. Dichas visualizaciones se encuentran en formato de video en la carpeta de Drive especificada en [29].

Luego, optamos por fijar la definición de *outlier* en el contexto que nos encontrábamos. Dado que en todos los casos las triangulaciones se trataban de estimaciones y no se contaba con la verdad absoluta de la posición del individuo, tuvimos que utilizar la definición de *outlier* desde un punto de vista estadístico [30]. Para ello se calcularon las distancias recorridas entre cada *frame* para cada *marker*. Después se ordenaron de menor a mayor y se encontró la mediana y el primer y tercer cuartil. Por último, con esos datos se definió que un *outlier* puede ser clasificado como tal si cumple con una de las siguientes condiciones:

$$\begin{aligned} D &< 1Q - 1.5M \\ D &> 1.5M + 3Q \end{aligned} \quad (7)$$

Donde *D* es la distancia, *1Q* es el primer cuartil, *M* es la mediana y *3Q* es el tercer cuartil.

Es interesante notar que, de todos los resultados obtenidos, la primera condición no arrojaba ningún *outlier*, es decir que todos los errores observados se pueden interpretar intuitivamente como que los métodos realizaban sobreestimaciones sobre las posiciones de los puntos (dicho de otra manera, los puntos se "movían de más").

Una vez definidos los *outliers* se procedió a estudiar los resultados de las diferentes implementaciones. Para ello se utilizaron tres *datasets* distintos, obtenidos de proyectos anteriores. Cada *dataset* surge a partir de la captura de una escena, cada una con un sujeto diferente, realizando distintos movimientos. En particular, en la primera escena analizada el sujeto se trata de la tutora de este proyecto, Marcela Alejandra Guerrero, mientras que en las escenas 2 y 3 se puede observar a los alumnos Braulio Sespede y Agustín Calatayud, respectivamente. A fin de que la comparación sea lo más imparcial posible, se decidió recortar la cantidad de *frames* de manera tal que todos los *datasets* tengan la misma cantidad. Por lo tanto, se tomó la más corta de las escenas, la cual resultó ser la escena 1, con aproximadamente 810 *frames* y se ajustó al resto acorde a ésta.

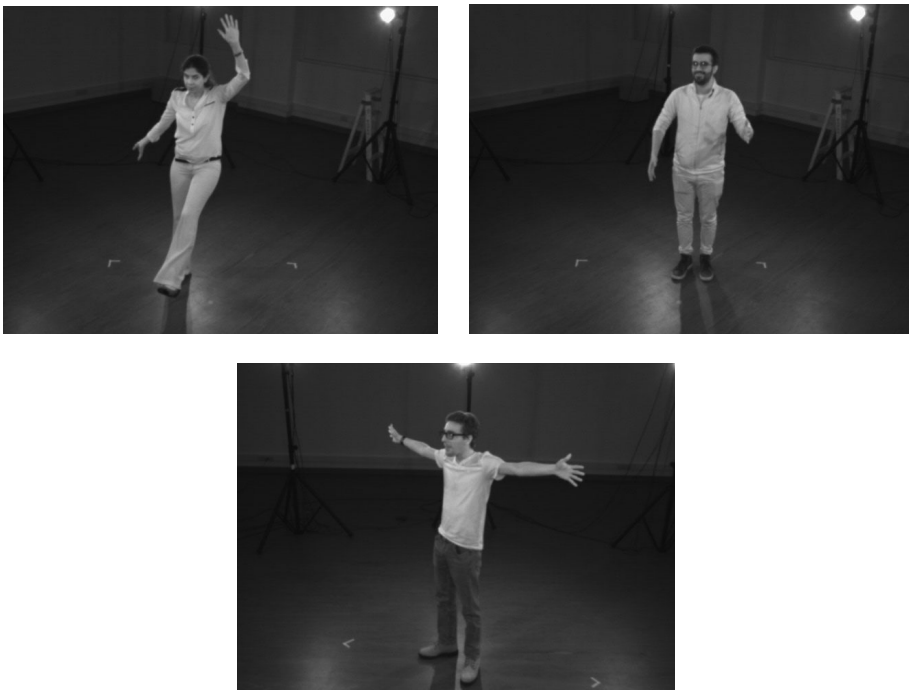

Figura 11: Capturas de las distintas escenas

Luego de ajustar los *datasets* se procedió a analizar en primer lugar la cantidad de *outliers* para la triangulación lineal implementada por *DLTdv*.

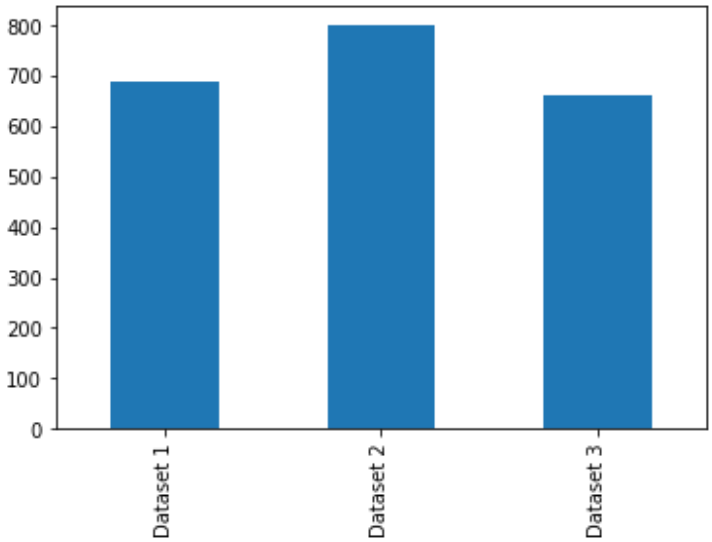

Figura 12: Cantidad de *outliers* para la triangulación lineal

Ya con los *outliers* identificados se realizó un cálculo para estimar la distancia promedio entre ellos, a lo que nosotros llamamos error. Podemos observar que para este método el error para los distintos *datasets* ronda entre 30 y 60 aproximadamente.

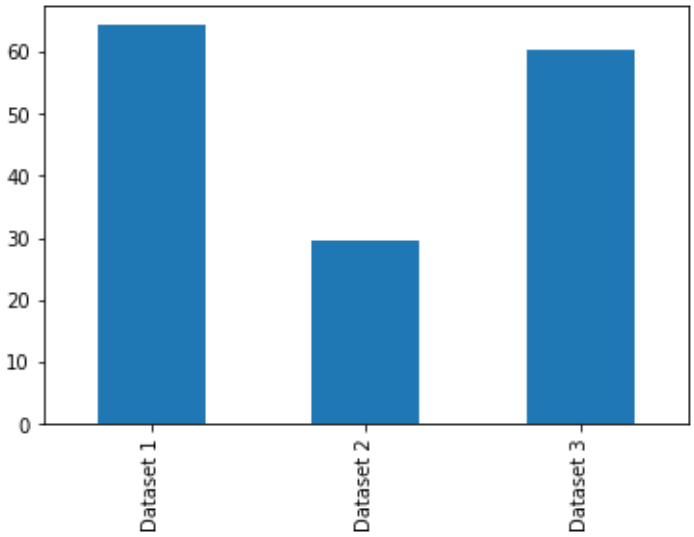

Figura 13: Error promedio para la triangulación lineal

En segundo lugar, se realizó el mismo análisis para la triangulación stereo, con y sin filtro, la cual arrojó los siguientes resultados en términos de cantidad de *outliers*.

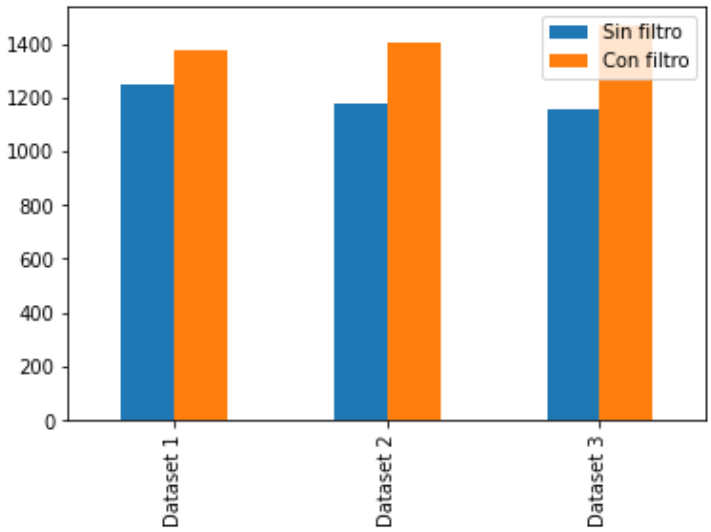

Figura 14: Cantidad de *outliers* para la triangulación stereo

Fue inesperado encontrar que la aplicación del filtro de suavización ocasionó la detección de más *outliers* ya que intuitivamente esperábamos lo contrario. Luego, para el mismo análisis de la triangulación stereo, con y sin filtro, se calculó también el error promedio.

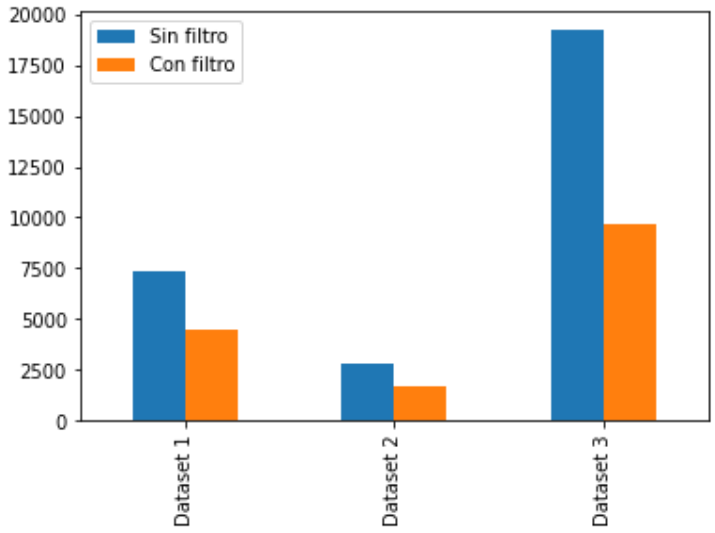

Figura 15: Error promedio para la triangulación stereo

Sin embargo, una vez que obtuvimos la figura 15 y observamos que el error era del orden de los miles y las decenas de miles, concluimos que el filtro puede estar comportándose anormalmente ya que no fue diseñado para corregir semejante magnitud de error.

Para la triangulación multi-cámara, se analizaron resultados con y sin filtro también. En particular, para los *outliers* se obtuvieron los siguientes datos:

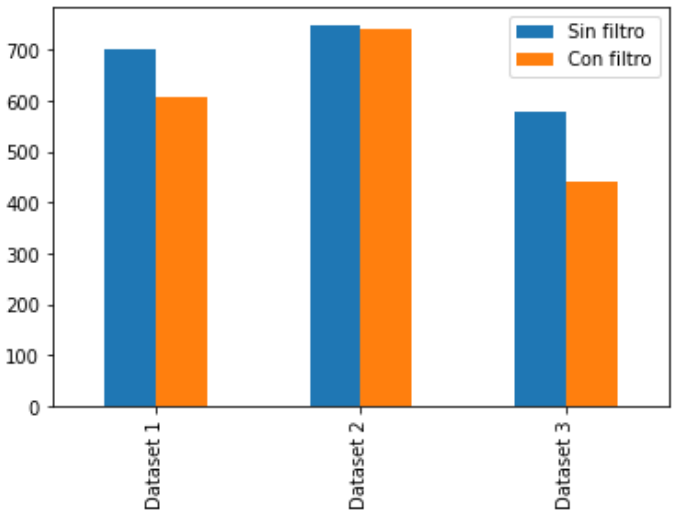

Figura 16: Cantidad de *outliers* para la triangulación multi-cámara

Se puede observar que la cantidad de *outliers* varía de manera similar a la triangulación lineal, donde se detectaron más *outliers* en el segundo *dataset*. En cuanto al error promedio, la triangulación multi-cámara dio los siguientes resultados:

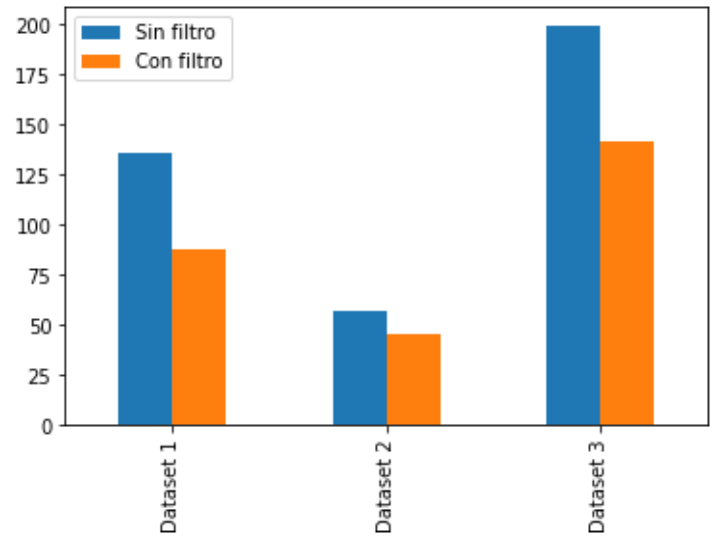

Figura 17: Error promedio para la triangulación multi-cámara

El error promedio tampoco muestra sorpresas. A diferencia de la triangulación stereo, ésta se comporta de manera esperada. Se puede observar con claridad que la aplicación del filtro logra el efecto deseado y, por ende, no sólo se reduce la cantidad de *outliers* sino que además el error medido es menor.

En la siguiente figura se presenta la cantidad de *outliers* para la triangulación de Anipose:

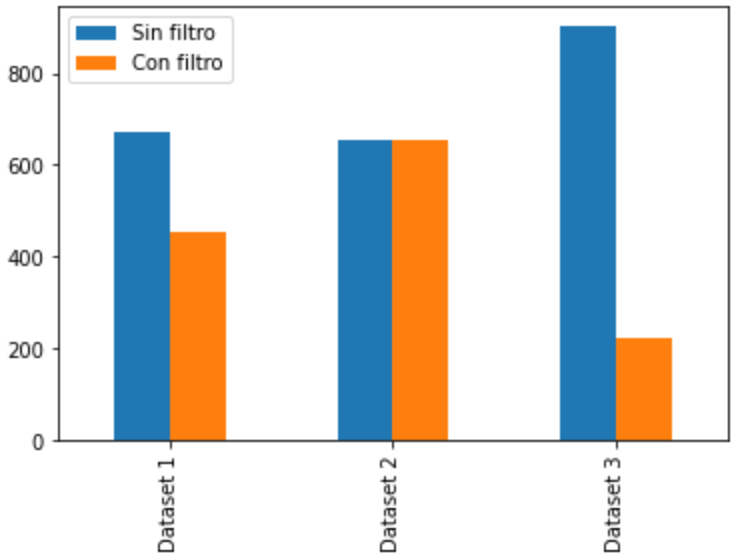

Figura 18: Cantidad de *outliers* para la triangulación de *Anipose*

Como se puede observar, una vez más el filtro resulta en la disminución de *outliers*, acorde a los resultados esperados originalmente. Es interesante notar que de todos los algoritmos estudiados que permiten la aplicación de filtros, en rasgos generales éste logró remover significativamente más *outliers* que sus pares.

Respecto al error promedio para la triangulación de *Anipose* se obtuvieron los siguientes datos:

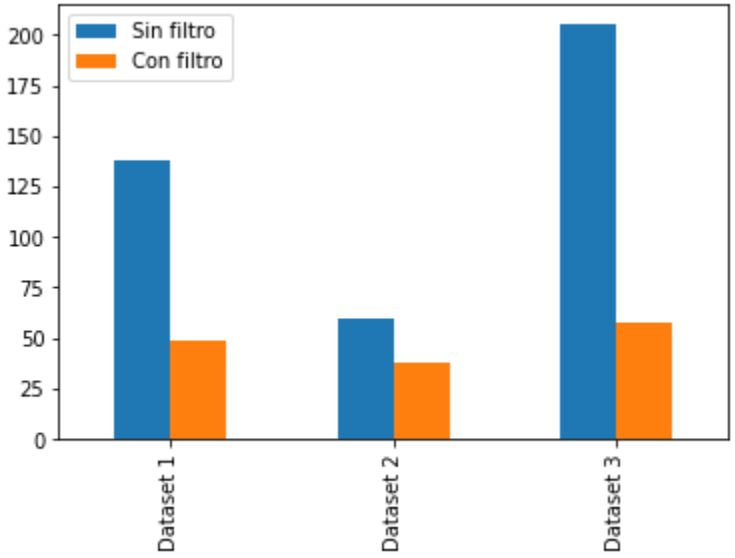

Figura 19: Error promedio para la triangulación de *Anipose*

Los datos de la figura 19 muestran un error promedio esperado acorde a los resultados obtenidos en las triangulaciones ya estudiadas.

Por último, una vez obtenidos todos los datos anteriores se decidió hacer una comparación tomando en cuenta los mejores resultados para cada tipo de triangulación. Por lo tanto, se estudiaron los casos de *DLTdv* y las triangulaciones multi-cámara y de *Anipose* con el filtro aplicado. En el caso de la triangulación stereo, se usaron los datos sin filtro para la comparación de *outliers* y los datos con el filtro aplicado para la comparación del error promedio. A continuación, se observa la cantidad de *outliers* para el conjunto recién descrito.

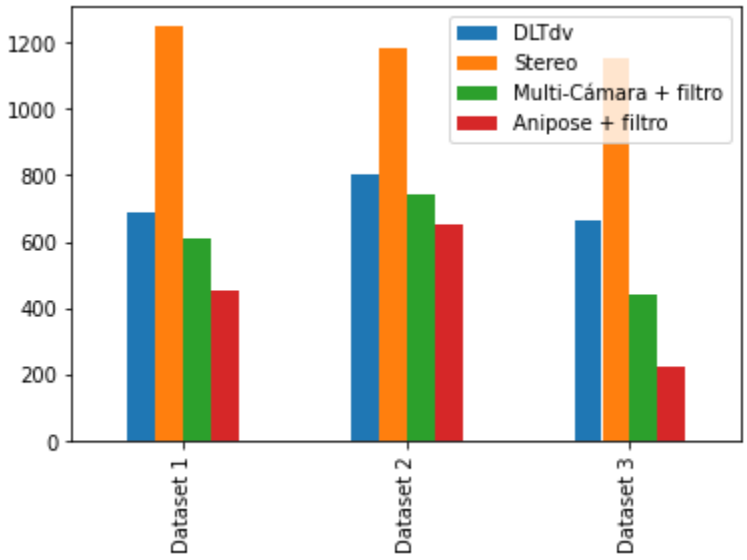

Figura 20: Comparación de *outliers* entre las distintas triangulaciones

Lo primero que se ve es que la triangulación stereo, obtiene la mayor cantidad de *outliers* en todos los *datasets* y que la triangulación de *Anipose*, cuando se le aplica el filtro, obtiene la menor cantidad de *outliers* para todos los *datasets*. El filtro de *Anipose* es una optimización diseñada específicamente para disminuir la cantidad de *outliers* dado que determina un umbral de error y elimina aquellos puntos que no son considerados suficientemente buenos. Esto quiere decir que elimina los *outliers* que detecta y los reemplaza por una interpolación. Como se puede observar, los resultados son muy buenos.

A continuación, se analiza el error promedio para los mejores resultados de cada triangulación para cada *dataset*.

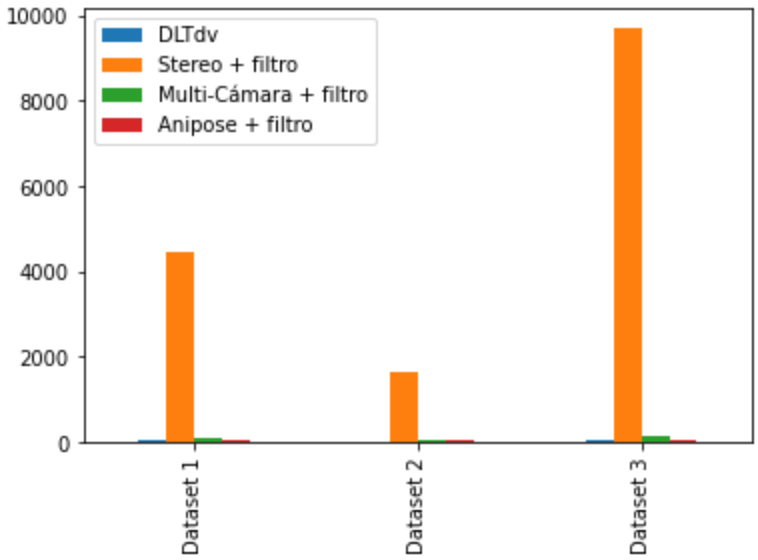

Figura 21: Comparación del error promedio entre las distintas triangulaciones

Como podemos observar en la figura 21, la triangulación stereo es considerablemente peor que el resto, no sólo arrojando más errores, sino que también dichos errores son mucho mayores y, por ende, más perceptibles al ojo humano. Por el otro lado, la mejor triangulación resultó ser la de *Anipose* (con el filtro aplicado) en ambas categorías, aunque los resultados son relativamente semejantes a los de *DLTdv* y la triangulación multi-cámara.

8. Conclusiones y Futuras Mejoras

En conclusión, más allá de las dificultades impuestas por la pandemia, fue posible estudiar los distintos métodos de triangulación tanto de manera teórica como sus diferentes implementaciones en *Matlab* o *Python*. Para ello fue necesario el uso de programas de preprocesamiento como *DeepLabCut* y el programa de calibración de las cámaras, así como programas de posprocesamiento para analizar los resultados.

En primera instancia, luego de visualizar las triangulaciones con *Ovito*, los estudios de *outliers* y errores pudieron corroborar nuestras ideas intuitivas sobre cuáles métodos ofrecían mejores resultados. La conclusión alcanzada sobre la poca precisión de la triangulación stereo fue que éste es el método que presenta más limitaciones, ya que busca utilizar pares de cámaras contiguos, pero no considera si los *markers* se encuentran ocluidos para dichos pares. En cambio, para el resto de las triangulaciones estimamos que la mayoría de los errores provienen de la imprecisión de los programas de preprocesamiento.

Habiendo dicho esto, consideramos que en otras circunstancias se podrían haber hecho estudios más exhaustivos. Por un lado, se podría haber probado con otras redes neuronales y de esta manera poder utilizar otro set de datos de entrada para los algoritmos de triangulación. Otra mejora podría haber sido la recalibración de las cámaras y el estudio de escenas donde las posiciones absolutas son conocidas, como, por ejemplo, mediante el uso de un *ChArUco board* en los videos utilizados para triangular. Esto hubiese sido especialmente útil para definir los errores de la triangulación de manera más precisa y sin tener que basarse en las definiciones estadísticas que fueron utilizadas. Sin embargo, debido al contexto de la pandemia y las restricciones que ésta conllevó, dichos análisis no fueron posibles de realizar. Vale aclarar que, si bien se contaba con los datos de una escena donde el *ChArUco board* estaba presente, la decisión de utilizar una red pre-entrenada con *markers* de humanos descartó la posibilidad de utilizar dicha escena para definir lo que se conoce como *ground truth* o información validada a través de observaciones y mediciones en vez de inferida. Es por esto que surge la necesidad de usar otro tipo de redes o de capturar más escenas, como fue mencionado anteriormente.

Para finalizar, se quisiera resaltar la importancia de conceptos aprendidos en materias como “Sistemas de Inteligencia Artificial”, “Métodos Numéricos” y “Métodos Numéricos Avanzados”, entre otras, para poder aplicar esos conocimientos en este trabajo. Además, fue necesario un arduo proceso de investigación y aprendizaje de manera individual y remota sobre temas de

computer vision y los distintos algoritmos de triangulación. Para ello fue indispensable la ayuda tanto de la tutora a cargo del proyecto, así como de los exalumnos que fueron mencionados, a quienes se les agradece enormemente.

9. Bibliografía y Referencias

[1] Hughes, J. F., McGuire, M. S., Foley, J., Sklar, D. F., Feiner, S. K., Akeley, K., Van Dam, A., & Foley, J. D. (2009). Computer graphics: Principles and practice (3rd ed.). Addison-Wesley Educational.

[2] Guerrero, M. A. et al. (2021) 3D Features Recovery from Images of a Multi Camera System. Poster Presentation. CSCE: The 2021 World Congress in Computer Science, Computer Engineering, & Applied Computing. Las Vegas, USA. Accepted for publication, to be published Dec 2021.

[3] Mikolás, L., & Calatayud, A. (2020). Markerless Human Tracking Report.

[4] Mathis, A., Mamidanna, P., Cury, K.M., Abe, T., Murthy, V.N., Mathis, M.W., & Bethge, M. (2018) DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nature Neuroscience. https://doi.org/10.1038/s41593-018-0209-y (Accedido por última vez: 22/07/2021)

[5] Sespede, B. (2019) 3D-pose: Optical multi-view motion capture with calibration suite. GitHub Repository. https://github.com/bsespede/3D-pose (Accedido por última vez: 22/07/2021)

[6] Robust line model estimation using RANSAC. https://scikit-image.org/docs/dev/auto_examples/transform/plot_ransac.html (Accedido por última vez: 14/08/2021)

[7] Ban, T. (2016) Polynomial-RANSAC. GitHub Repository. https://github.com/tituszban/Polynomial-RANSAC (Accedido por última vez: 14/08/2021)

[8] Hartley, R., & Zisserman, A. (2004). Multiple View Geometry in Computer Vision (2nd ed.). Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511811685 (Accedido por última vez: 27/07/2021)

[9] Torr, P. H. S., & Zisserman, A. (2000). Feature Based Methods for Structure and Motion Estimation. https://www.doi.org/10.1007/3-540-44480-7_19 (Accedido por última vez: 16/08/2021)

[10] Denker, K., Lehner, B. & Umlauf, G. (2011). Real-time triangulation of point streams. Engineering with Computers. https://doi.org/10.1007/s00366-010-0181-y (Accedido por última vez: 16/08/2021)

[11] Cyrill, S. (2021). Online Training: Photogrammetric Computer Vision, YouTube. https://www.youtube.com/playlist?list=PLgnQpQtFTOGTPQhKBOGgjTgX-mzpsOGOX (Accedido por última vez: 11/08/2021)

[12] Simek, K. (2013). Intrinsic Camera Parameters. *Sightations, A computer vision blog*, Github. http://ksimek.github.io/2013/08/13/intrinsic/ (Accedido por última vez: 28/07/2021)

[13] Simek, K. (2012). Extrinsic Camera Parameters. *Sightations, A computer vision blog*, Github. http://ksimek.github.io/2012/08/22/extrinsic/ (Accedido por última vez: 28/07/2021)
[14] Hartley, R., & Sturm, P. (1997). Triangulation. https://perception.inrialpes.fr/Publications/1997/HS97/HartleySturm-cviu97.pdf (Accedido por última vez: 12/08/2021)
[15] Mathis, A., Mamidanna, P., Cury, K.M., Abe, T., Murthy, V.N., Mathis, M.W., & Bethge, M. (2018). DeepLabCut Model Zoo. http://www.mackenziemathislab.org/dlc-modelzoo (Accedido por última vez: 30/07/2021)
[16] Cámara Flex 3 Optitrack; https://optitrack.com/cameras/flex-3/indepth.html (Accedido por última vez: 27/08/2021)
[17] Kwon, Y. (1998). Direct Linear Transformations. http://www.kwon3d.com/theory/dlt/dlt.html (Accedido por última vez: 30/07/2021)
[18] Hedrick, T. L. (2021) DLTdv digitizing tool. *MATLAB tools for digitizing video files and calibrating cameras*. https://biomech.web.unc.edu/dltdv/ (Accedido por última vez: 30/07/2021)
[19] Karashchuk, P. (2019). Anipose Documentation. *Zenodo*. https://anipose.readthedocs.io/en/latest/index.html (Accedido por última vez: 30/07/2021)
[20] Karashchuk, P. (2019). Aniposelib Repository. Github. https://github.com/lambdaloop/aniposelib (Accedido por última vez: 30/07/2021)
[21] Bradski, G. (2000). The OpenCV Library: Camera Calibration and 3D Reconstruction. *Dr. Dobb's Journal of Software Tools*. https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (Accedido por última vez: 29/07/2021)
[22] Mikolás, L., & Calatayud, A. (2020). Markerless Human Tracking: Repositorio utilizado de sistema de captura de markers sin traje. Github. https://github.com/acalatayud/Markerless-Human-Tracking/ (Accedido por última vez: 29/07/2021)
[23] Straw, A. (2013). PyMVG Documentation. https://pymvg.readthedocs.io/en/latest/api_reference.html (Accedido por última vez: 30/07/2021)
[24] Hedrick, T. L. (2008). Software techniques for two- and three-dimensional kinematic measurements of biological and biomimetic systems. https://iopscience.iop.org/article/10.1088/1748-3182/3/3/034001/meta (Accedido por última vez: 13/08/2021)
[25] Hedrick, T. L. (2020). Dltdv Repository. GitHub. https://github.com/tlhedrick/dltdv (Accedido por última vez: 13/08/2021)
[26] Karashchuk, P., Rupp, K. L., Dickinson, E. S., Walling-Bell, S., Sanders, E., Azim, E., Brunton, B. W., & Tuthill, J. C. (2020). Anipose: a toolkit for robust markerless 3D pose

estimation. Cold Spring Harbor Laboratory. https://doi.org/10.1101/2020.05.26.117325 (Accedido por última vez: 13/08/2021)
[27] Karashchuk, P. (2019). Anipose Repository. GitHub. https://github.com/lambdaloop/anipose (Accedido por última vez: 13/08/2021)
[28] Stukowski, A. (2010). Visualization and analysis of atomistic simulation data with OVITO – the Open Visualization Tool Modelling Simul. Mater. Sci. Eng. 18, 015012. https://www.ovito.org/ (Accedido por última vez: 05/08/2021)
[29] Resultados en formato de video; https://tinyurl.com/triangulaciones-videos (Accedido por última vez: 05/08/2021)
[30] Renze, J. "Outlier." From MathWorld--A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/Outlier.html (Accedido por última vez: 05/08/2021)